

BACKLOG DO SOFTWARE GRIPY

ADRIANO PAULO LAES DE SANTANA

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

MACAÉ - RJ
SETEMBRO - 2019

BACKLOG DO SOFTWARE GRIPY

ADRIANO PAULO LAES DE SANTANA

Relatório apresentado ao Grupo de Inferência em Reservatório (GIR) contendo a lista detalhada de tarefas a serem executadas, e seu grau de realização, no desenvolvimento futuro do software griPy.

MACAÉ - RJ
SETEMBRO - 2019

Lista de ilustrações

Figura 1 – Exemplo de janela responsável pela seleção dos dados de um “DataFilter” usado em objeto 5-D, no caso um dado sísmico “aberto” decomposto no domínio da frequência. Figura extraída de um vídeo utilizado na apresentação do trabalho de Santana (2017) .	5
Figura 2 – Janela “Importar como” do LAS Loader.	8
Figura 3 – Janela “WellPlotEditor”, com destaque para seleção de escala de eixo X pelo usuário.	10
Figura 4 – Janela “WellPlotEditor”, com destaque para seleção de cores Matplotlib pelo usuário.	11
Figura 5 – Janela de propriedades para a projeção em gráfico de uma curva sintética de <i>Well Log</i> (UIBaseObject).	12
Figura 6 – Janela de propriedades para uma curva sintética de <i>Well Log</i> (DataObject).	12

Sumário

1	BACKLOG GRIPY - SETEMBRO 2019	1
1.1	Curvas de índice (IndexCurve)	1
1.1.1	Grau de realização	2
1.1.2	Solução adotada: DataIndex e DataIndexMap	2
1.2	Redefinição de objeto “DataFilter”	3
1.2.1	Grau de realização	5
1.2.2	Solução adotada	5
1.3	Reescrita de classes “DataTypes”	6
1.3.1	Grau de realização	6
1.3.2	Solução adotada	6
1.4	Finalização do design integrado dos Canvas para plotagem das curvas	6
1.4.1	Grau de realização	7
1.4.2	Solução adotada	7
1.5	Reescrita de Classes de Plotagem de curvas e dados	7
1.5.1	Grau de realização	7
1.5.2	Solução adotada	7
1.6	Revisão dos conceitos “Tipo de curva” e “Tipo de dado” nos perfis de poços	8
1.6.1	Grau de realização	9
1.6.2	Solução adotada	9
1.7	Modelo de visualização/edição de propriedades de TODOS objetos griPy	9
1.7.1	Grau de realização	11
1.7.2	Solução adotada	11
1.8	Reescrita de algumas janelas do software	13
1.8.1	Grau de realização	14
1.9	Remodelagem da árvore de objetos	14
1.9.1	Grau de realização	14
1.9.2	Observação sobre a tarefa	14
	REFERÊNCIAS	15

1 Backlog griPy - setembro 2019

Após a defesa do trabalho proposto por [Santana \(2017\)](#), o projeto do software griPy teve seu foco na conversão do código fonte para a versão 3 da linguagem de programação Python (griPy3), para os ajustes necessários às novas versões do sistema de janelas wxPython e para alguns importantes ajustes na leitura e escrita de projetos griPy (arquivos .pgg) pois o pacote Numpy sofreu algumas mudanças que, apesar de não muito divulgadas por sua comunidade, causaram impacto significativo ao griPy.

No segundo semestre do ano de 2018, este autor (que atualmente se considera líder de produto do griPy) se dedicava a execução de algumas tarefas¹. Na sequência, já no ano de 2019, os esforços foram concentrados na resolução das tarefas que seguem neste relatório. Neste período, a equipe do GIR passou a contar com novos profissionais, que serão capazes de dar sequência ao trabalho e manter o nível de qualidade, além possibilitar o vislumbre de novas oportunidades ainda não exploradas.

Atualmente, o GIR tem percebido a necessidade de equalizar as informações entre o desenvolvimento prévio realizado no software griPy, feito nos últimos 3 anos de forma praticamente integral pelo autor deste, e a equipe que passou a integrar o projeto recentemente. Assim, por sugestão do professor Fernando Moraes que é o líder do projeto griPy, é apresentado nesta obra um *backlog*, ou seja uma lista detalhada das tarefas, que do ponto de vista deste autor, devem nortear este importante projeto.

As seguintes pendências estão elencadas por ordem de prioridade, da mais prioritária para a de menor prioridade, não necessariamente significando que a execução da primeira tarefa trará um ganho maior ao software do que a execução da última.

Esta atualização, datada de setembro 2019, aponta o grau de desenvolvimento de cada uma das tarefas propostas no mês de maio. As soluções adotadas também estão descritas, quando julgou-se necessário.

1.1 Curvas de índice (IndexCurve)

Na fase inicial de desenvolvimento do griPy, optou-se em utilizar o conceito de uma única curva de eixo Z (profundidade/tempo) para todos os perfis de um determinado poço. A IndexCurve era associada ao objeto da classe Well, passando a

¹ Conforme constam no arquivo *changelog* da versão Python 3 do griPy.

servir de referência a todos os perfis do poço em questão. Nas primeiras versões do software optou-se em implementar esta ideia, pois o conceito de poço foi fortemente influenciado pela estrutura de um arquivo LAS.

Ao observar outras ferramentas de mercado, que servem de inspiração para o desenvolvimento do griPy (tais como IP, Teclog e ferramentas de uso interno da Petrobras) percebe-se que a curva de profundidade/tempo é associada diretamente a cada uma das curvas de perfis. A evolução do software levou ao desenvolvimento de outras classes, inclusive com objetos multidimensionais. Um dos exemplos que mais ilustram as limitações (e trabalhos adicionais necessários) é quando se transforma um perfil para o domínio da frequência. Apesar do objeto 2-D criado (frequências e profundidade/tempo) o eixo Z está atrelado diretamente ao objeto poço. Nesse código então, esse objeto 2-D realiza uma “pergunta” ao perfil para saber quem é seu `IndexCurve`, que por sua vez realiza a mesma “pergunta” ao objeto poço.

Assim, recomenda-se que as curvas de eixo Z sejam associadas diretamente aos objetos filhos, mesmo que isso venha a gerar uma duplicidade de dados nas curvas de profundidade/tempo.

1.1.1 Grau de realização

Tarefa concluída (100%).

1.1.2 Solução adotada: `DataIndex` e `DataIndexMap`

Optou-se em renomear a classe `IndexCurve` para `DataIndex`. Em conjunto, foi dado um maior grau de independência a relação dos `DataObjects` com suas respectivas `DataIndexes`.

Desta forma, foi criada a classe `DataIndexMap` para correlacionar os índices de um determinado `DataObject`. Cada `DataObject` possui somente 1 `DataIndexMap`, e esse é capaz de “listar” os índices do `DataObject`.

Adicionalmente, foi incluído um método *wrapper* para que o desenvolvedor não necessite chamar diretamente a criação do `DataIndexMap`. Abaixo segue a descrição deste método:

```
def _create_data_index_map(self, *args):  
    """  
    args: List of lists of DataIndex uids.  
  
    Examples:
```

```

1-D data: [index.uid, owt_index.uid, twt_index.uid]

5-D data: [i_line_index.uid],
          [x_line_index.uid],
          [offset_index.uid],
          [freq_index.uid],
          [time_index.uid, prof_index.uid]
"""

```

Um exemplo de uso pode ser observado no código abaixo:

```

well = OM.new('well', name='NOME POCO')
OM.add(well)
curve_set = well.create_new_curve_set()
index = OM.new('data_index', name=index_name, datatype=datatype,
              unit=unit, start=start, samples=samples, step=ts)
OM.add(index, curve_set.uid)
log = OM.new('log', data, name='NOME LOG', unit='amplitude',
            datatype='SYNTHETIC')
OM.add(log, curve_set.uid)
log._create_data_index_map([indexuid])

```

1.2 Redefinição de objeto “DataFilter”

Considerando a possibilidade de se trabalhar no griPy com classes de dados (DataTypes) multidimensionais, classes estas que foram necessárias à confecção do projeto de dissertação do autor ([SANTANA, 2017](#)), foi criada a primeira versão da classe “DataFilter”².

Apesar do modelo primário de “DataFilter” apresentado outrora, seu conceito deve ser o mesmo: ser uma espécie de View/Query para que se trabalhe somente com a massa de dados necessária à aplicação em um determinado momento. Isto traz funcionalidade e desempenho ao software. O “DataFilter” em um dado 1-D atua somente como um seletor de informações. Por exemplo, deseja-se trabalhar com um perfil somente em uma determinada região do poço ou em uma camada geológica. Este objeto então seria o responsável esta seleção (em sua obtenção) no dado

² “DataFilter” é mencionado entre aspas, pois não se sabe se esta nomenclatura será mantida.

original. Ressalta-se que uma mesma seleção proveniente de um mesmo “DataFilter” pode ser aplicada a vários Tracks ou CrossPlots, por exemplo.

O dado selecionado por esta View/Query estará instanciado em memória somente, e ao final do processo desejado poderá ser salvo na base de dados (projeto) do software, como por exemplo um Zona em um poço.

Por sua vez, o “DataFilter” em duas ou mais dimensões possui um conceito mais amplo, pois além de selecionar o dado como no caso supracitado, ele atua como um seletor de dimensões do dado a serem exibidas e um ordenador dessas dimensões. Como um exemplo, é possível considerar um gather convertido em um domínio da frequência (vide dissertação Adriano Santana). Este é um objeto de 3 dimensões: ângulo/offset, frequência e profundidade/tempo. Neste exemplo, é possível plotar este dado em um plot 3-D, em um plot 2-D com um ângulo/offset pré definido e todo o range de frequências ou em um plot 2-D com uma frequência pré definida e todo o range de ângulos/offsets. O objeto responsável por esta seleção é o “DataFilter”.

A seguir, a figura 1 ilustra a representação de um “DataFilter” através do sistema de janelas do software griPy.

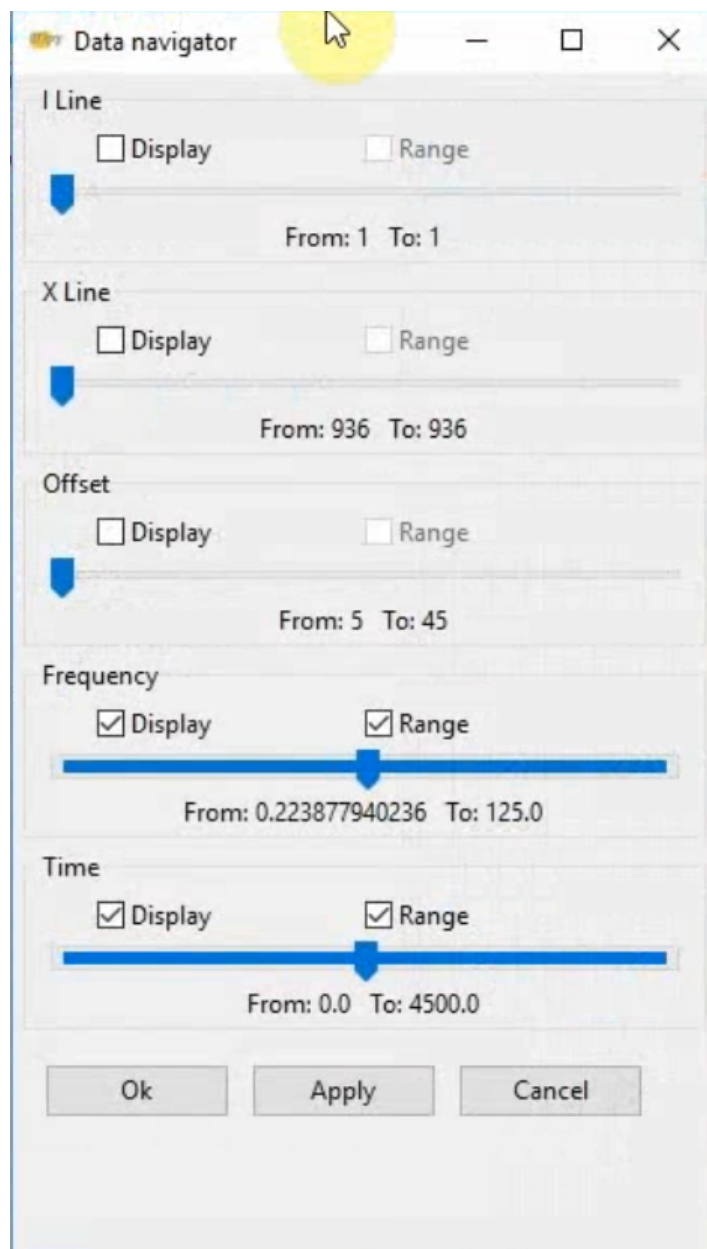


Figura 1 – Exemplo de janela responsável pela seleção dos dados de um “DataFilter” usado em objeto 5-D, no caso um dado sísmico “aberto” decomposto no domínio da frequência. Figura extraída de um vídeo utilizado na apresentação do trabalho de [Santana \(2017\)](#).

1.2.1 Grau de realização

Tarefa concluída (100%).

1.2.2 Solução adotada

A classe então chamada de DataFilter foi absorvida pela classe TrackObjectController.

1.3 Reescrita de classes “DataTypes”

No novo conceito do griPy3, as classes que anteriormente eram chamadas de classes “DataTypes”³ agora não recebem uma distinção como outrora. Assim, o fato do objeto possuir ou não uma propriedade “data” é, de certa forma, considerado um por menor.

Entende-se que a reescrita dessas classes é dependente da remodelagem do “DataFilter”.

1.3.1 Grau de realização

Tarefa concluída (100%).

1.3.2 Solução adotada

As classes necessárias ao desenvolvimento das tarefas propostas neste relatório foram todas reescritas, conforme proposto. Demais classes a serem adicionadas ao projeto devem ser herdeiras de DataObject e seguir as recomendações de uso constantes nas 2 primeiras seções deste trabalho.

1.4 Finalização do design integrado dos Canvas para plotagem das curvas

Foi realizado um amplo redesign nas classes que servem de base para o desenho das curvas no griPy. Essas classes são herdeiras diretas das classes FigureCanvas do Matplotlib. Neste processo foram unificados conceitos e propriedades das classes Tracks e CrossPlots, considerando que em boa parte essas classes são similares. Para atingir este objetivo, foi criada uma classe “mãe” intermediária que reúne as propriedades e funções comuns a Tracks e CrossPlots.

A importância desse trabalho poderá ser observada no futuro, pois este tipo de estrutura permitirá a criação de qualquer tipo de plots em 2 dimensões.

Esta tarefa já está, em grande parte, realizada. Porém sua conclusão depende da remodelagem de “DataFilter” e de “DataTypes”, para que então sejam realizados os testes necessários e os eventuais ajustes.

³ “DataTypes” é apresentado entre aspas por, nesta data, ser um nome obsoleto.

1.4.1 Grau de realização

O design dos canvas dos objetos de Track de CrossPlots (ou simplesmente Plots) estão integrados. Porém, o uso do CrossPlot necessita de alguns ajustes. Realização: 90%.

1.4.2 Solução adotada

Foi adotada a solução proposta inicialmente.

1.5 Reescrita de Classes de Plotagem de curvas e dados

Após a remodelagem de “DataFilter”, de “DataTypes” e também da conclusão do redesign dos Canvas, será necessário reescrever as classes de plotagem de curvas e dados (constantes no arquivo `ui/mvc_classes/track_object.py`). Essas classes são as responsáveis por representar os dados plotados sobre os Canvas. Em geral, as classes de objetos gerenciadas pelo ObjectManager possuem ao menos uma representação de plotagem.

O redesign dessas classes permitirá uma plotagem muito mais rápida, “clean” e dinâmica. Uma das possibilidades, por exemplo, é a criação de Zonas (topo e base) em um determinado poço de forma dinâmica (com o clicar do mouse para marcar topo e base).

Ressalta-se a importância da criação de Zonas não somente para o trabalho de interpretação e edição de perfis de poços, mas também como “suporte”, quando considerado o conceito de Inferência Bayesiana (pacotes de classes Rocha).

1.5.1 Grau de realização

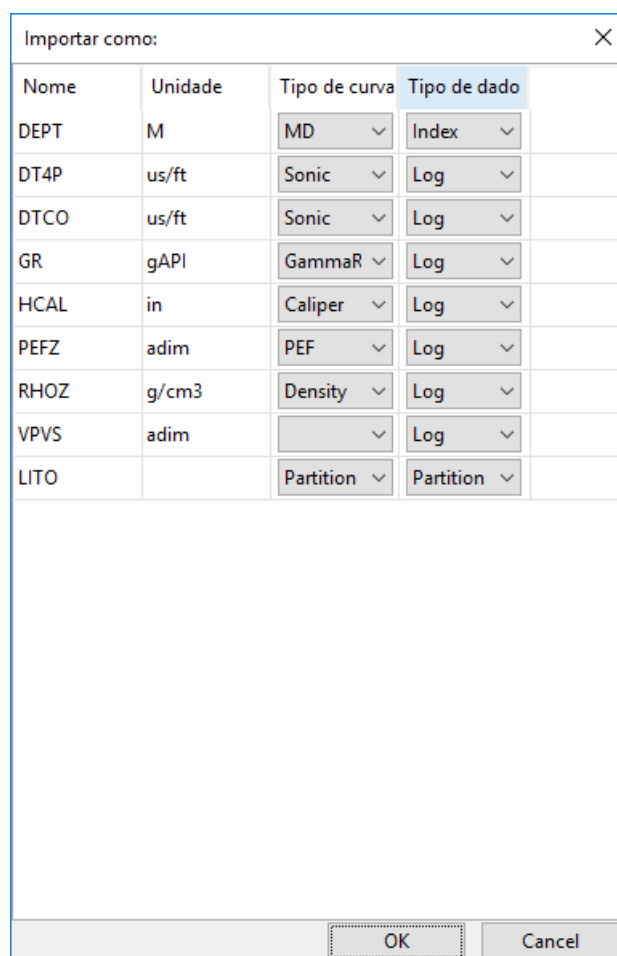
Tarefa concluída (100%).

1.5.2 Solução adotada

As classes foram revistas conforme proposto inicialmente. As representações por linha, index e por densidade foram separadas em arquivos individuais. Foram feitos ajustes nos TrackLabels para melhor identificar as curvas.

1.6 Revisão dos conceitos “Tipo de curva” e “Tipo de dado” nos perfis de poços

Conforme pode ser observado na janela “Importar como” do LAS Loader (vide figura 2), os tipos de curvas e tipos de dados necessitam de uma revisão/hierarquização interna (no arquivos da pasta basic/parms). Propõe-se a revisão das associações entre os tipos de dados e os tipos de curvas, bem como a exclusão dos tipos de curvas que não forem mais necessários ao griPy.



Nome	Unidade	Tipo de curva	Tipo de dado
DEPT	M	MD	Index
DT4P	us/ft	Sonic	Log
DTCO	us/ft	Sonic	Log
GR	gAPI	GammaR	Log
HCAL	in	Caliper	Log
PEFZ	adim	PEF	Log
RHOZ	g/cm3	Density	Log
VPVS	adim		Log
LITO		Partition	Partition

OK Cancel

Figura 2 – Janela “Importar como” do LAS Loader.

Neste processo, cabe a reflexão se deve-se manter o sistema de votos para as curvas não identificadas ou se este sistema deve ser readequado. De qualquer forma, essas alterações devem estar em conformidade com a escrita adotada para a classe Log (responsável pelos perfis).

1.6.1 Grau de realização

Tarefa concluída (100%).

1.6.2 Solução adotada

O ParametersManager foi revisado e os conceitos “Tipo de curva” e “Tipo de dado” foram ajustados. Foi mantido o sistema de votos para a dos tipos envolvidos.

1.7 Modelo de visualização/edição de propriedades de TODOS objetos griPy

Ao ter expandido o sistema de Messaging (pacote pypubsub) para todas as propriedades de todos objetos griPy3 (no mesmo modelo que as propriedades das classes herdeiras de UIModel eram na versão griPy 0.8 - Python 2.7) se tornou possível apresentar ao usuário o estado de qualquer objeto griPy em uma forma de “Grid de Propriedades” (baseado em wx.propgrid.PropertyGrid).

Para isto, é necessário o desenvolvimento de um Wrapper que correlacione os tipos de propriedades com seus respectivos objetos wx.propgrid.PGProperty. Esse desenvolvimento (e também dos demais wrappers) visa a facilitar o trabalho dos demais programadores (*maindevs* e de *plugins* do software).

Um exemplo é a propriedade “x_scale” da classe LineRepresentationController (arquivo ui/mvc_classes/track_object.py) que possui a seguinte codificação na versão griPy3:

```
_ATTRIBUTES['x_scale'] = {
    'default_value': 0,
    'type': int,
    'pg_property': 'EnumProperty',
    'label': 'X axis scale',
    'options_labels': ['Linear', 'Logarithmic'],
    'options_values': [0, 1]
}
```

No exemplo acima, a opção “pg_property” é a responsável por sinalizar para o Wrapper qual tipo de wx.propgrid.PGProperty exibirá o valor da propriedade ao usuário, no caso um EnumProperty. A descrição da propriedade para o usuário é definida em “label”. Por sua vez, “options_labels” apresenta as opções disponíveis

para o usuário no ComboBox responsável pela seleção. Finalizando, “options_values” descreve os valores associados a cada uma das possíveis escolhas.

Considerando esse exemplo, “Linear” retornará para propriedade “x_scale” o valor 0 enquanto a seleção de “Logarithmic” retornará o valor 1. A figura 3 apresenta o resultado do código acima quando processado pelo Wrapper.

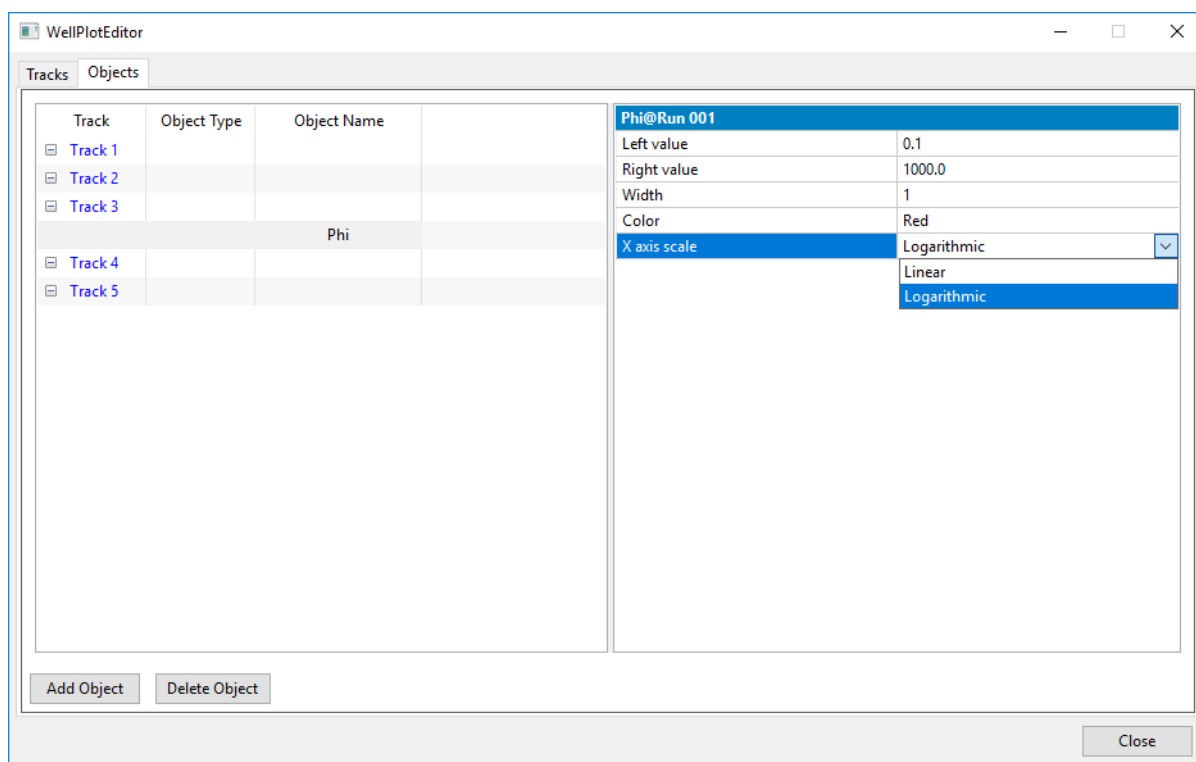


Figura 3 – Janela “WellPlotEditor”, com destaque para seleção de escala de eixo X pelo usuário.

O Wrapper de exibição de propriedades também permite pré fabricar alguns importantes exibidores/seletores que tendem a aparecer por diversas vezes no griPy. Um exemplo é o seletor de cores Matplotlib, conforme código abaixo que também pertence a mesma classe da propriedade “x_scale”.

```
_ATTRIBUTES['color'] = {
    'default_value': 'Black',
    'type': str,
    'pg_property': 'MPLColorsProperty',
    'label': 'Color'
}
```

O código acima é apresentado ao usuário conforme ilustra a figura 4.

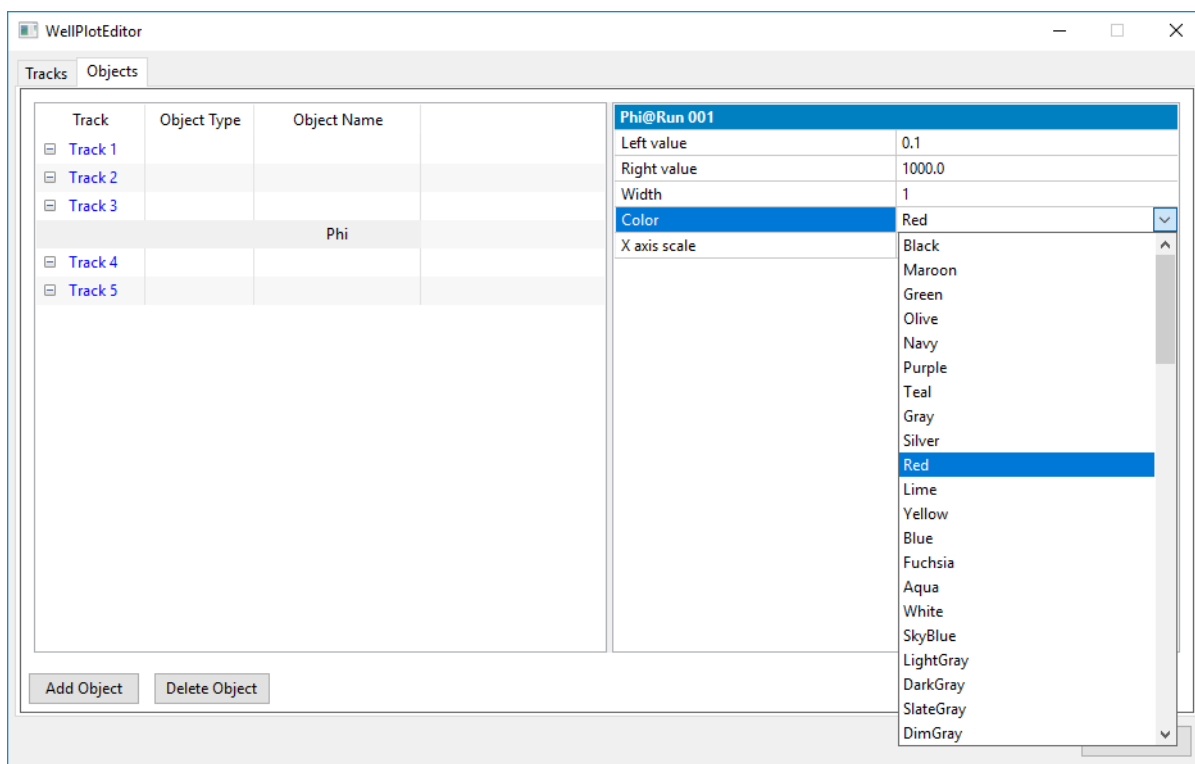


Figura 4 – Janela “WellPlotEditor”, com destaque para seleção de cores Matplotlib pelo usuário.

O autor deste relatório considera que esta tarefa e a reconstrução do “DataFilter” (seção 1.2) são as mais importantes e as que mais representarão para o futuro do griPy.

1.7.1 Grau de realização

Tarefa concluída (100%).

1.7.2 Solução adotada

Foi criada uma estrutura que permite visualização e edição de propriedades de objetos do GriPy, sejam eles DataObjects ou UIObjects (UIBaseObjects). As figuras 5 e 6 ilustram o sistema criado.

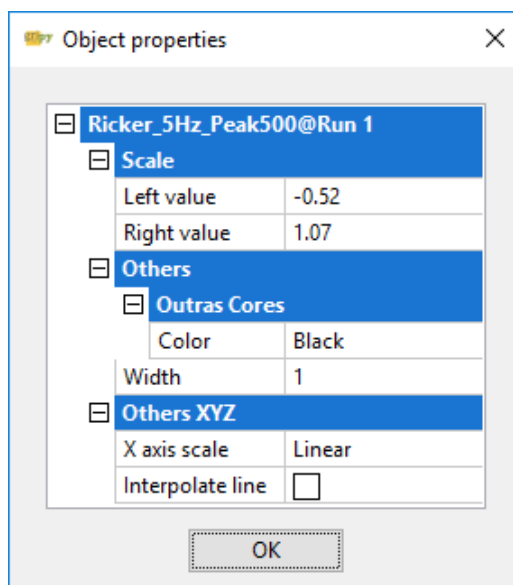


Figura 5 – Janela de propriedades para a projeção em gráfico de uma curva sintética de *Well Log* (UIBaseObject).

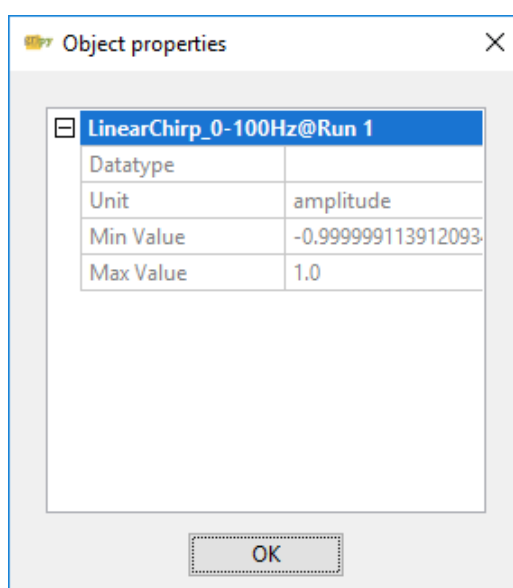


Figura 6 – Janela de propriedades para uma curva sintética de *Well Log* (DataObject).

Abaixo tem-se um pequeno exemplo, extraído da classe `CanvasPlotterController`. Nele, as categorias e as propriedades exibidas nas figuras anteriores são assim definidas:

```
def _get_pg_categories(self):
    cats = OrderedDict([
```



```
        ('category_general', OrderedDict([
            ('label', 'General')
        ])),
    ])
    return cats

def _get_pg_properties(self):
    props = OrderedDict()
    props['rect'] = {
        'pg_property': 'StringProperty',
        'label': 'Rect',
        'category': 'category_general'
    }
    props['xscale'] = {
        'label': 'X axis scale',
        'pg_property': 'MPLScaleProperty',
        'category': 'category_general'
    }
    props['yscale'] = {
        'label': 'Y axis scale',
        'pg_property': 'MPLScaleProperty',
        'category': 'category_general'
    }
```

1.8 Reescrita de algumas janelas do software

Percebe-se que algumas importantes janelas do griPy necessitam ser atualizadas para o sistema de Janelas criado para o griPy (wrapper wxPython):

- LAS Loader (após revisão/hierarquização dos tipo de curvas e tipo de dados), vide figura 2;
- WellPlotEditor, vide figuras 3 e 4;
- Debug Console;
- etc.

1.8.1 Grau de realização

Tarefa concluída (100%).

1.9 Remodelagem da árvore de objetos

Questiona-se o fato da exibição do projeto griPy em árvore ser necessariamente uma réplica do ObjectManager. Considera-se a possibilidade e a importância de um objeto “saber como se exibir” ou não, na árvore de objetos.

Esta mudança facilitaria a visualização do projeto griPy por parte do usuário, evitando possível poluição na interface (usabilidade).

1.9.1 Grau de realização

Tarefa parcialmente concluída (40%).

1.9.2 Observação sobre a tarefa

Considera-se que esta tarefa não possui importância significativa, quando comparada às demais. Há o entendimento por parte deste autor de que o atual estágio de desenvolvimento da árvore de objetos é capaz de atender as demandas necessárias ao software. Assim, optou-se em sua não conclusão neste momento.

Referências

SANTANA, A. P. L. de. *Desenvolvimento de software e análise multiespectral da resposta sísmica em reservatórios de hidrocarbonetos*. Tese (Dissertação) — Universidade Estadual do Norte Fluminense, 2017. Citado 4 vezes nas páginas ii, 1, 3 e 5.